SEMAINE D'ETUDE MATHÉMATIQUES ET ENTREPRISES, CIRM - MARSEILLE 14–18 AVRIL 2014

TESTING THE RELIABILITY OF A TRUE RANDOM GENERATOR AT RUN TIME

FLORIAN CAULLERY[(1)], ALEXANDER GETMANENKO[(2)], VITO MANDORINO[(3)], AND VINCENT VONG[(4)]

Abstract. We tackle the problem of deciding at run time if the output of a true random generator seems reliable or not. We must consider the context, namely the limited available memory and the need to answer yes or no in a very short time. We explore three points of view: a Boolean functions based approach by the way of the study of the non-linearity, an algebraic and computational approach by the way of sub word complexity and then a spectral approach by the way of Fourier transform and statistical analysis of the Fourier transformed sequence by the Shapiro–Wilk test.

Contents

Date: June 8, 2014.

Key words and phrases. randomness, true random generator.

The authors thank Yannick Teglia, Senior Member of Technical Staff of STMicroelectronics, for proposing this subject.

The project was supervised by Robert Rolland Associated Senior Researcher at Institut de Mathém atiques de Marseille.

1. Introduction - Industrial context

1.1. **True random number generators.** In various algorithmic applications, there arises a need for sequences of numbers with good properties of randomness. Often, as in many programming languages, an algorithm producing pseudo random sequences is sufficient. For more advanced purposes such as information security, an analogue (as opposed to purely digital) hardware unit generating random numbers is attached to the processor. The resulting sequence of numbers possesses randomness due to resumed thermal white noise present in a simple physical system; such a sequence is not predictable algorithmically in principle and therefore is called *a true random number sequence*.

Applications of true random number generators (TRNG) include generation of cryptographic keys and Counter-measures against Physical Attacks (e.g. Side Channel Attacks, Fault Injection).

An important task is therefore to be able to diagnose whether a given hardware unit generating random numbers functions properly, which means that the random numbers "behave as a sequence independent identically distributed random variables" – an expression whose formal meaning will be addressed later in this report.

Of particular concern for the needs of cryptographic security is a possibility that the TRNG is put under a physical attack to become more predictable, e.g. it is exposed to a laser field, electromagnetic injection, alpha particles, or is malfunctioning due to e.g. power supply or clock glitches. Predictability of a TRNG may lead to vulnerability to cryptographic keys that it generates.

Let us remark that the random number generated by the TRNG are post-processed digitally to make any malfunctioning of the TRNG as little exploitable as possible. It is the testing at run time of the quality of random numbers before any digital post-processing takes place that will be the subject of this report.

1.2. **Mathematical problem.** Imagine a small TRNG unit inside a small electronic device (think: cell phone) that needs to quickly diagnose itself before sending an encrypted message. A relatively short sequence of numbers may be generated, there are limitations of memory and processing time, and the cost of a false alarm is potentially the need to replace a part of a cell phone.

Obviously, a sequence that contains too many consecutive 0's or too many consecutive 1's should be rejected, as well as sequences with strong periodic patterns such as 010101010..., as well as sequences where the portion of 1's is significantly different from 50 percent. Other possible tokens of malfunctioning TRNG have been studied, e.g., in [1]. Theoretical developments with respect to this problem include concepts of complexity theory, pseudo-randomness of finite sequences, numerical evaluation of Kolmogorov of short strings, non-linearity of Boolean functions.

The enterprise has quantified the constraints as follows:
– one needs to make a yes/no decision with respect to proper functioning of a TRNG based on a string of up to 512 bit (0 or 1) that it has generated.

– 2 kb - 512 bits of memory available for running computations
– programs and relevant constants can be stored in a different, larger memory

Further, we have concentrated on detecting periodic regularities in the random sequence, which would correspond to exposure of the TRNG to a periodic perturbation (e.g., by a periodic electromagnetic field).

Rather then trying to write a new chapter in the theory of randomness, we have tested various approaches similar to the existing ones but with a focus on the above mentioned concern and constraints.

2. An algebraic point of view

2.1. **Boolean functions based approach.** Let $\mathbb{F}_2 \cong (\mathbb{Z}/2\mathbb{Z}, +, \times)$ be the finite field with two elements and $\mathbb{F}_2^n$ be the vector space of dimension n over $\mathbb{F}_2$. A function f from $\mathbb{F}_2^n$ to $\mathbb{F}_2$ is called a Boolean function. Such functions are entirely determined their truth table and admit a representation as a polynomial in $\mathbb{F}_2[x_1, \ldots, x_n]$ which is called the *algebraic normal form of* f.

One can see a sequence of length $l = 2^n$ as the truth table of a Boolean function. Hence every sequence define a unique Boolean function and our problem is transformed into deciding if a Boolean function is random. There exists various indicators on a Boolean functions. We have selected two of them to measure their level of "randomness": the *non-linearity* and the *absolute indicator*. The decision to consider the non-linearity was motivated by the work of Schmidt and Rodier on the asymptotic distribution of non-linearity of random Boolean function [7, 8]. On the other hand, there is no theoretical background on the distribution of the absolute indicator of random Boolean functions and the decision to consider the absolute indicator entirely relied on numerical experiments.

2.1.1. *Non-linearity.* Before defining the non-linearity of a Boolean function we need to define the Hamming distance of two functions and the affine functions.

Definition 2.1. Let f and g be two Boolean functions from $\mathbb{F}_2^n$ to $\mathbb{F}_2$. The Hamming distance $d(f, g)$ between f and g is defined as:

$$d(f, g) = \sum_{x \in \mathbb{F}_2^n} f(x) + g(x).$$

This distance is actually the number of time where $f(x)$ and $g(x)$ have a different output.

Definition 2.2. A function $A : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ is said affine if the total degree of the algebraic normal form of A is at most one.

The non-linearity of a Boolean function is defined as follow:

Definition 2.3. The non-linearity $NL(f)$ of a Boolean function is the Hamming distance between f and the set of affine functions:

$$NL(f) = \min_{A \text{ affine}} (A, f)$$

To compute the non-linearity of a function, we use the following formula.

Proposition 2.4. *We have:*

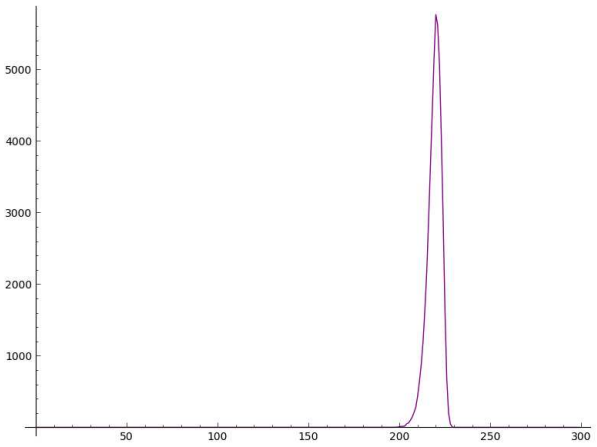
$$NL(f) = 2^{m-1} - \frac{1}{2} \sup_{v \in \mathbb{F}_2^n} \left| \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)+v.x} \right|.$$

The computation of the non-linearity can be done in a linear time $0(l)$ where $l = 2^n$ is the length of the sequence. The details of the algorithm can be found in http://robert.rolland.acrypta.com/telechargements/algebre/Hadamard.pdf

In our study, we focused on sequences of length $2^9 = 512$. We computed the non-linearity of $50,000$ random Boolean functions and obtain the following table and graph:

Mean	219.1854
Median	220
Variance	15.1837
Standard deviation	3.8966

Numerical experiments on non-linearity



Non-linearity distribution

From this experiments, we could deduce the following test:

(1) Consider a sequence of length 512 as the truth table of $f : \mathbb{F}_2^9 \mapsto \mathbb{F}_2$
(2) calculate the non-linearity of f
(3) if it is between 215 and 225 keep it
(4) don't use it otherwise

This test enabled us to discard sequences with a period up to 260.

2.1.2. *Absolute indicator.* The absolute indicator of a Boolean function f is defined as the maximum absolute value of its auto-correlation function. The auto-correlation function measure the similarities of f with f composed with a shift.

Definition 2.5. Let $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be a Boolean function. The auto-correlation function AC_f of f is defined as

$$AC_f(a) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)+f(x+a)}.$$

Definition 2.6. Let $f : \mathbb{F}_2^n \mapsto \mathbb{F}_2$ be a Boolean function. The absolute indicator $AI(f)$ of f is defined as

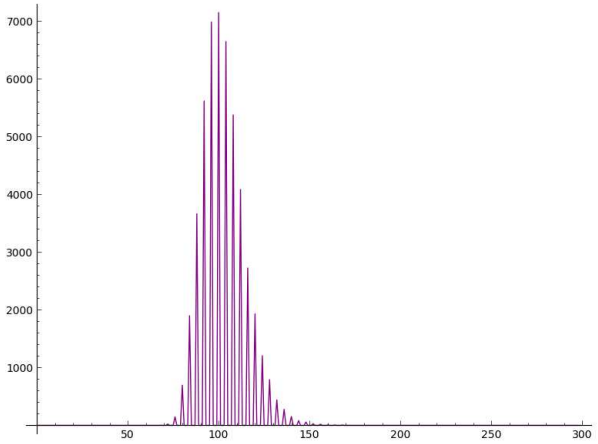
$$AI(f) = \max_{a \in \mathbb{F}_2^n - \{0\}} AC_f(a).$$

The computation of absolute indicator is made in $O(2^{2n})$ operation where 2^n is the length of the sequence.

Again, we focused on sequences of length $2^9 = 512$. We computed the absolute indicator of $10,000$ random Boolean functions and obtain the following table and graph:

Mean	100.0625
Median	100
Standard deviation	24.1242

Numerical experiments on absolute indicator



Absolute indicator distribution

From this experiments, we could deduce the following test:

(1) Consider a sequence of length 512 as the truth table of $f : \mathbb{F}_2^9 \mapsto \mathbb{F}_2$
(2) calculate the absolute indicator of f
(3) if it is between 85 and 115 keep it
(4) don't use it otherwise

This test enabled us to discard sequences with a period up to 280.

2.1.3. *Summary.* The second test we proposed seems a bit better since it discards sequences with longer period but it is also slower. One shall also remark that the non-linearity can be obtain from the computation of the absolute indicator without additional operation. Hence, the combination of the two tests could be a good option if the microchip resources allow it. Otherwise, the non-linearity test seems the best option with this approach.

2.2. **Algebraic and computational approach.** One of the good notions in order to measure if a sequence is random, is the Kolmogorov complexity. But this quantity is not computable. So, instead of it, we can use other complexity which can be computed, in order to have an approximation of the notion "be random". In our context, we just want to reject two types of binary sequences:

- the periodic ones,
- the linear Boolean functions.

The method is classical: we have to find some functions on words, such that they have an expected behavior on random words, but not for periodic and linear ones.

One of the given approach is based on a simple notion, the sub word complexity [2]. The another one will be based on the computation of the Lempel-Ziv

complexity [5]. In the sequel, a sequence of bits is seen as a word over the alphabet $\{0,1\}$.

2.2.1. *Sub word complexity and pseudo-randomness.* Let $w = w_1 w_2 \cdots w_n$ be a word over $\{0,1\}$. Let us define the sub word complexity of a word.

Definition 2.7. Let f and w be two finite words. The word f is a ***factor*** of w if there exists two words u and v such that $w = ufv$.

Example 2.8. If $w = 1011011$, then $f = 1101$ is a factor of w, since $w = 10 \cdot f \cdot 1$.

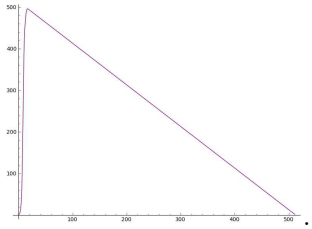
Definition 2.9. Let w be a finite word. The ***subword complexity*** of w is the number of the different factors of w. We denote by $CF(w)$ the sub word complexity of w, by $CF_k(w)$, the number of factors of size k of w, and by $Factor_k(w)$ the set of factors of size k of w.

Example 2.10. If $w = 01001101$, then:

$$Factor_4(w) = \{0100,\ 1001,\ 0011,\ 0110,\ 1101\},$$

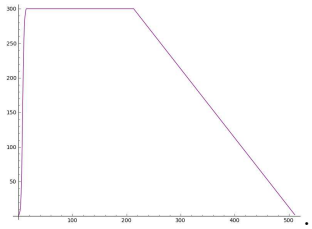
and $CF_4(w) = 5$.

Let w be a random word of size 512, and let us consider the finite sequence $(CF_k(w))_{k \leq 512}$. By computer test, we observe the following graph:



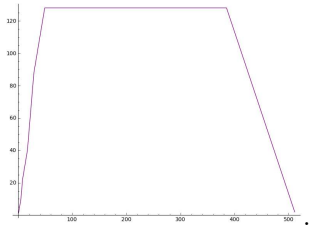
.

For w a random 300 periodic binary word of size 512, here is the behavior:



.

For w a random linear Boolean function of size 512, here is the behavior:



.

By heuristic, we directly see that by studying the sequence $(CF_k(w))$, we can determine a form of randomness. As we can observe, only the first terms of the sequence are important, in order to discriminate if a sequence is random. Then, we deduce the following algorithm:

(1) calculate the first terms of the factor complexity (≤ 30)
(2) if the behavior is similar to the random one, keep it
(3) don't use it otherwise.

Here is some test for words of length 512:

k	CF_k of a LBF	CF_k of 70 PW	CF_k of 150 PW	CF_k of 400 PW	CF_k of RW
3	6	8	8	8	8
5	10	32	32	32	32
7	14	58	91	124	124
15	30	70	150	400	493
25	66	70	150	400	488
30	86	70	150	400	483

LBF: linear Boolean functions. PW: periodic words. RW: random words.

As we can see, for a random word w of size 512, generally $CF_{25}(w)$ is greater than 450, but for "special" words, is lesser than 400. So we have a good test in order to reject periodic words, and linear Boolean functions.

2.2.2. *The Lempel-Ziv complexity.* Complexity theory is also intimately related to the compression one. Indeed, in this context, a random word is word which compression rate is "bad". Since the Lempel-Ziv complexity is related to a compression algorithm, one can test if this notion gives some results in our problem. Let us recall some definitions, in order to compute the Lempel-Ziv complexity for a word.

Definition 2.11. Let $w = w_1 w_2 \cdots w_n$ be a finite word. We denote by $w(i,j)$ the factor $w_i \cdots w_j$. The ***exhaustive history*** of w is the factorization withe the most factors of $w = f_1 \cdots f_k$, such that for $i < k$, f_i is a word satisfying the following properties :

(1)
- if we set $f_i = a_1 \cdots a_j$ (a_i is a letter), then f_i is not a factor of $w' = f_1 f_2 \cdots f_{i-1} a_1 a_2 \cdots a_{j-1}$,
- but $a_1 a_2 \cdots a_{j-1}$ is a factor of $v = f_1 f_2 \cdots f_{i-1} a_1 a_2 \cdots a_{j-2}$.

The factor f_k satisfies the second property.

The factors f_i are the components of the exhaustive history, and the Lempel-Ziv complexity of w is the number k, denoted by $LZ(w)$.

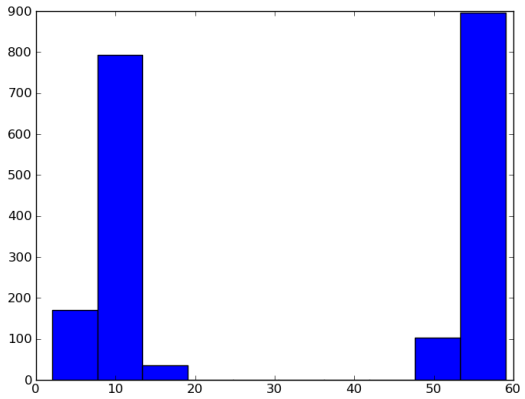
The following algorithm determines the exhaustive history of a word $w = w_1 \cdots w_n$:

- initialization: $f_1 = w_1$
- if we have find f_1, f_2,...,f_i, we have $w = f_1 \cdots f_i v$. Then, we search the shortest proper prefix p of v satisfying the conditions (1). If p exists, then $f_{i+1} = p$. Otherwise, $f_{i+1} = v$, and the algorithm is finished.

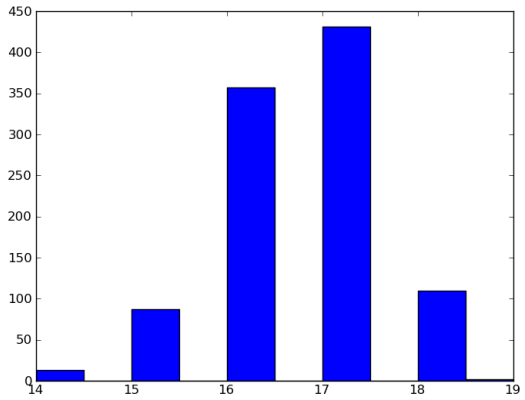
Example 2.12. If we set $w = 0010100111000$, we have:

- initialization: $f_1 = w_1 = 0$. Then, $w = 0 \cdot 010100111000$.
- 01 is the shortest proper prefix of 010100111000, satisfying (1). Then, $f_2 = 01$, and $w = 0 \cdot 01 \cdot 0100111000$.
- By the same arguments, we have $f_3 = 0100$, $f_4 = 11$, and $f_5 = 1000$.
- Finally, $w = 0 \cdot 01 \cdot 0100 \cdot 11 \cdot 1000$, and $LZ(w) = 5$.

In the following histogram, we represent the Lempel-Ziv complexity of random binary sequences of size 512, and linear Boolean functions of same size:



As we can see, for a random sequence, the value is between 50 and 60. But, for a linear Boolean function, the value is lesser than 20. For one thousand 100-periodic word, we have the following histogram:



So, we deduce that a binary word of size 512 may be random if its Lempel-Ziv complexity is between 50 and 60. By the criteria, we can see that linear Boolean function and small periodic word are not random.

3. Identification of deviance of a random number generator - Spectral approaches

As the device generating random numbers is likely to be subjected to a *periodic* electromagnetic field due to either an influence of the environment, or by malignant attack, methods based on the Fourier transform (FT) appear to be a natural choice for detecting such perturbation. Such methods are referred to as “spectral methods".

The work that we accomplished and that we are reporting in this section, can be summarized as follows:

Given the goal of detecting *periodic perturbations*,
the size of the data 512 bit,
and memory constraints $\sim$ 2Kb,
we have performed a **comparative study of various methods inspired by Fourier transform.**

There is a whole family of such methods differing by:

- a choice of a Fourier transform among various versions, and
- a choice of a statistical criterion for quantifying anomalies in the resulting FT coefficients.

3.1. **Discrete Fourier Transform and Fast Fourier Transform.** Our starting point was the paper [1] where the authors propose to use the **discrete Fourier transform** (DFT) technique to analyze for periodic patterns. Let us recall the definition:

Definition. For a sequence $v_0, ..., v_{N-1}$ of complex numbers, define

$$F_j = \sum_{k=0}^{N} v_k e^{2\pi\sqrt{-1}kj/N} \in \mathbb{C}. \tag{2}$$

For the sake of exposition and intuition, it is more convenient to encode the string of random bits $v_0, .., v_{N-1}$ as ± 1 rather than 0 and 1. Since all v_j are real, the resulting sequence F_j will have the following symmetry:

$$F_j = \overline{F_{N-j}}, \tag{3}$$

thus effectively containing only N real numbers.

DFT gives a measure of several periodic components in a discrete sequence, see [1, Sec.3.6] and figures therein.

While a straightforward implementation of the formula (2) has time complexity $O(N^2)$, the *Fast Fourier Transform* (FFT) algorithm reduces the complexity to $O(N \log N)$.

To briefly review the idea of the FFT algorithm following [4, Ch.8], let us rewrite the formula (2) in the matrix notation as

$$\mathbf{F} = \mathbf{A}\mathbf{v}$$

and suppose for simplicity that column vectors $\mathbf{F}$ and $\mathbf{v}$ have length 2^γ. Here, the matrix $\mathbf{A}$ has entries

$$\mathbf{A}_{jk} = (\zeta^{j\times k})_{j,k}, \qquad \zeta = e^{2\pi i/N}.$$

Morally (see [4] for a precise statement), the matrix $\mathbf{A}$ can be represented as a product

$$\mathbf{A}_{(\log_2 N)}...\mathbf{A}_{(1)}$$

where matrices $\mathbf{A}_{(k)}$ are very sparse (and, of course, independent of the vector $\mathbf{v}$). Multiplication by a constant sparse matrix requires the number of operations equal to the number of its nonzero entries; $\log_2 N$ matrix multiplication by matrices each containing $O(N)$ nonzero entries gives the times complexity of $O(N \log N)$ of the FFT algorithm.

Let us now analize the memory needs of the FFT algorithm. Let us write

$$\mathbf{F}_{(0)} = \mathbf{v}$$
$$\mathbf{F}_{(j)} = \mathbf{A}_{(j)}\mathbf{F}_{(j-1)}$$

Assuming that the matrices $\mathbf{A}_{(j)}$ are pre-calculated and stored in the permanent memory as a part of the algorithm, at each step of the algorithm described in [4] we need to store vectors $\mathbf{F}_{(j)}$ and $\mathbf{F}_{(j-1)}$ which makes the total of $2N$ complex numbers, or $4N$ real numbers, plus a few loop counters etc. We expect that symmetry reduces this to $2N$ real numbers.

Let us elaborate a little on the point of symmetry of the complex vectors $\mathbf{F}_{(j)}$ originating from the fact that the initial vector $\mathbf{v}$ is real. At this point we are not ready to give a precise general statement, but we expect that it may be known to specialists. In a particular example of $N = 8$, i.e. $\gamma = 3$, and using [4, eqs (8.40)-(8.43)], we have in our notation that

$$\mathbf{F}_{(0),k} \in \mathbb{R}, \quad k = 0, ...7$$
$$\mathbf{F}_{(1),k} \in \mathbb{R}, \quad k = 0, ...7$$
$$\mathbf{F}_{(2),k} \in \mathbb{R}, \quad k = 0, ...3; \quad \mathbf{F}_{(2),4} = \overline{\mathbf{F}_{(2),k+2}}, \quad k = 4, 5$$

and $\mathbf{F}_{(3),k}$ has symmetries corresponding to (3). For a particular N that will be used in an implementation of the algorithm, an *ad hoc* description of the symmetries will suffice.

In the particular case of $N = 512$, allocating 4 bytes per real number, the FFT algorithm will require 4 Kilobytes plus a few bytes to store loop counters etc. This is twice as much as is available according to the constraints set by the industrial partner; one solution may be to reduce N to 256 (this approach will be analyzed below in sec.3.6), another – to implement real numbers using 2 bytes only which may be feasible as the real numbers in question are of approximately the same order of magnitude and as the precision-sensitive operation of the division of reals never enters the algorithm.

3.2. **Statistical analysis of the Fourier transformed sequence by the Shapiro–Wilk test.** It is well-known, or at least widely accepted, see [1] that if $v_n \in \{\pm 1\}$ are independent random variables, then $\operatorname{Re} F_n$, $\operatorname{Im} F_n$ are independent normal random variables with mean 0 and the standard deviation $\sqrt{\frac{N}{2}}$:

$$\operatorname{Re} F_n,\ \operatorname{Im} F_n \ \sim\ \mathcal{N}\left(0, \sqrt{\frac{N}{2}}\right).$$

Several tests for normality of a sample are available in the literature: Anderson-Darling test, Lilliefors test, Kolmogorov-Smirnov test, Pearson's chi-squared test, and the test that we have chosen based on its good reputation – the Shapiro-Wilk test. With respect to other tests, the Shapiro-Wilk test has the lowest rate of false negatives and it performs well for recommended for sample size up to 3000 – the range that includes our sample size of 512.

Let us briefly review the Shapiro-Wilk test following [3]. The basic idea is to compare the sorted sample with an expected sorted sample from the normal distribution. Starting with a sorted sample of real numbers $\mathbf{y} = (y_1, ..., y_N)$, $y_1 \le ... \le y_N$, the test is based on the quantity

$$W = \frac{(\sum_{i=1}^{N} a_i y_i)^2}{\sum_{i=1}^{N} (y_i - m)},$$

where

$$m = \operatorname{mean} \mathbf{y},$$

and the coefficients a_i are independent of the sample, well-studied real constants (see [6]) that can be stored as a part of the algorithm. The quantiles of the test quantity for a given sample size are also well-known.

The R language provides a function `shapiro.test`. The calculation of W involves sorting of N numbers, which takes time of $O(N \log_2 N)$, while for $N = 512$ the memory of 1 Kb will clearly be enough.The sum of N real numbers takes time of order $O(N)$ and a negligible and constant amount of memory.

3.3. **Selection of test examples and confidence level.** To evaluate and compare performance of different methods of detecting periodic regularities in a sequence of random numbers, we have generated perturbed random sequences according to the formula

$$\mathbf{v}_n = sign(\mathcal{N}(0,1) + b * \sin(kn)) \tag{4}$$

where $\mathcal{N}(0,1)$ are pseudo random numbers, normally distributed around the origin with unit variance, generated by the programming language R, and

$$b = 0,\ 0.01,\ 0.02,\ \ldots\ 2.00 \quad \text{strength of the perturbation}$$

$$k = 0,\ 0.1,\ \ldots, 3.2 \quad \text{frequency of the perturbation}$$

Note this choice includes perturbations with non-integer period.

We will reject a sequence as non-random if the P-value of a given statistical test is < 0.001.

3.4. **Implementing the FFT + Shapiro-Wilk test.** We implemented our algorithm in the R language. Both Fast Fourier transform and the Shapiro-Wilk test are available as standard functions of the language. Here is the code for the algorithm.

```
1
2  # assigning the length of our sequences
3
4  len = 512
5
6  # initialize the vector of interest which counts the number of strings
       accepted as random
7  # for a given perturbation strength b. We use 201 possible values for
       b.
8
9  count = matrix(0, nrow = 201, ncol = 1)
10
11 # looping on different values of the perturbation strength b  and of
       the frequency j.
12 # For each couple of values (b,j) we repeat the test 15 times
13
14 for(b in 0:200){ for (j in 0:32){ for (h in 1:15){
15   sequence = sign(rnorm(len)+(b/100)*cos((j/10)*c(0:(len-1))))
16 #generate a pseudo-random sequence according to formula (4)
17   coeffs = fft(sequence) #use the Fast Fourier Transform to get the
         Fourier coefficients of the sequence
18   SW_Data = c(Re(coeffs[1:(floor(len/2)+1)]), Im(coeffs[2:floor(len/2)
         ]))
19 #get rid of the symmetries (3)
20 #to get the vector which is to be fed to the Shapiro-Wilk test
21   SW_Pval = shapiro.test(SW_Data)$p.value
```

```
#get the p-value given by the Shapiro-Wilk test. We are going to
    accept the sequence as random
#if its p-value is >= 0.001
  count[b+1] = count[b+1] + (SW_Pval >= 0.001)
#count the number of accepted strings under perturbation strength  b
}}}

percent = count * (100/33*15)  #convert counting to percentages for
    plot purposes

plot(percent)
```

.

One of the runs of the algorithm gave the data in fig. 1.

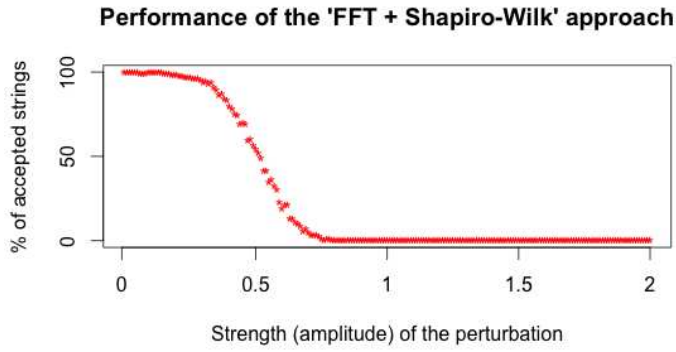


FIGURE 1. *Performance of the Discrete Fourier Transform followed by the Shapiro-Wilk test. Here the horizontal axis corresponds to the strength of the periodic perturbation b as in* (4).

We can see that for values of the strength parameter b in (4) which are below approximately 0.2 the test does not detect any deterioration in the quality of the random bits, while almost all strings generated with $b \geq 0.7$ are rejected as non-random at significance level 0.001.

3.5. **Variant #1: Hadamard-Walsh Transform.** Having proposed and tested one possibility for the detection of periodic perturbations, we continued with exploration of various variants of this approach.

The first variant is to run the Shapiro-Wilk test on the data coming not from the Discrete Fourier Transform, but rather from the Hadamard-Walsh transform (HWT) of the sequence of bits $\mathbf{v}$.

In case of strings of bits of length $N = 512$, the Hadamard-Walsh transform can be thought of an analogue of the DFT with the group $\left(\frac{\mathbb{Z}}{2\mathbb{Z}}\right)^9$ replacing $\frac{\mathbb{Z}}{512\,\mathbb{Z}}$ Indeed, the DFT described above is equivalent to calculating

$$\mathbf{F}_\chi = \sum_k \mathbf{v}_n \chi(n)$$

for every character χ of the discrete group $\mathbb{Z}/512\mathbb{Z}$

$$\chi \ : \ \mathbb{Z}/512\mathbb{Z} \ \to \ \mathbb{C}.$$

Hadamard transform repackages the 512 numbers $\mathbf{v}_n$ as parametrized by $\{0,1\}^9$. The HWT is thus calculated by the formula

$$\mathbf{H}_\chi = \sum_n \mathbf{v}_n \chi(n) \tag{5}$$

for every character χ of the group $(\mathbb{Z}/2\mathbb{Z})^9$:

$$\chi \ : \ (\mathbb{Z}/2\mathbb{Z})^9 \ \to \ \{\pm 1\} \subset \mathbb{C}.$$

There is an analogue of the FFT algorithm for the HWT with the complexity of $O(N \log N)$. However, the advantage of the HWT is that it computes with integers only which saves time and relieves the memory shortage difficulty that we encountered in the analysis of the usual DFT algorithm, page 10. One may expect however that a method based on the HWT will heavily stress periods of length 2^k.

An experimental comparison of the methods based on FFT and HWT, in both cases followed by the Shapiro-Wilk test with significance level 0.001, is presented on figure 2.

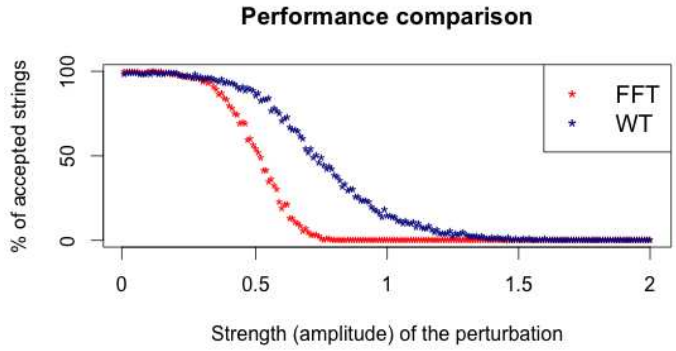


FIGURE 2. *Comparison between the FFT + Shapiro-Wilk test (red curve) and HWT + Shapiro-Wilk test (blue curve), same axes as on fig. 1*

Here is the R code for this method. The Hadamard-Walsh transform is available in the package `boolfun`.

```
len = 512
count = matrix(0, nrow = 201, ncol = 1)

for(b in 0:200){ for (j in 0:32){ for (h in 1:15){
  sequence = (sign(rnorm(len)+(b/100)*cos((j/10)*c(0:(len-1)))) >= 0 )
#we go back to sequences of 0's and 1's since this is the proper
    setting for HWT
  SW_Data = walshTransform(sequence)
  SW_Pval = shapiro.test(SW_Data)$p.value
  count[b+1] = count[b+1] + (SW_Pval >= 0.001)
}}}

percent = count * (100/33*15)
plot(percent)
```

.

3.6. **Variant #2: halving the string length.** As the FFT algorithm proposed earlier struggles with memory constraints, see page 10, we have also considered and evaluated the possibility to start with only 256-bit strings of 0's and 1's. Halving the input data size naturally leads to somewhat lower performance, fig 3.

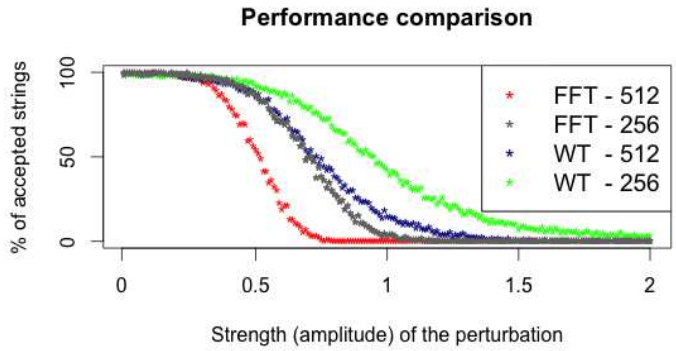


FIGURE 3. *FFT + Shapiro-Wilk test for 512-bit and 256-bit strings, respectively, as well as HWT + Shapiro-Wilk test for 512-bit and 256-bit strings.*

3.7. **Variant #3: Hadamard-Walsh Transform + non-linearity test.** The Non-linearity test discussed in the framework of algebraic methods, sec. **PUT A CROSS-REFERENCE HERE!**, fits in the picture of the spectral methods as follows.

For a sequence of $\mathbf{v}_{n=0,\dots,N-1}$ with $N = 2^\gamma$, the non-linearity can be equivalently defined as

$$N(\mathbf{v}) = \max_{\chi:(\mathbb{Z}/2\mathbb{Z})^\gamma \to \mathbb{C}} |\mathbf{H}_\chi|$$

where the maximum is taken over all characters of the group $(\mathbb{Z}/2\mathbb{Z})^\gamma$ and $\mathbf{H}_\chi$ is the Hadamard-Walsh transform as in (5).

Unusually low values of non-linearity is a sign of low quality of generated random numbers.

To our knowledge, the distribution of $N(\mathbf{v})$ is not yet fully understood theoretically, so we have calculated the histogram of its distribution in case $N = 512$ by using the R pseudo-random numbers, fig. 4

We found experimentally the 0.001-quantile to be 203, thus in our test we are going to reject a string as non-random if its non-linearity is ≤ 203.

The advantage of non-linearity is that it can be calculated within the time $O(N)$ working entirely with integers; however the first order statistics is a much cruder test of normality than the more expensive Shapiro-Wilk test. The performance of Non-linearity test compared to other tests presented earlier is given on figure 5, followed by the related R code which uses the `nl` function from the `boolfun` package.

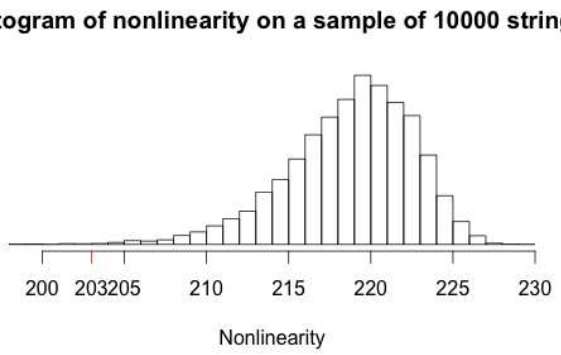


FIGURE 4. *The experimental histogram for the distribution* $N(\mathbf{v})$ *provided the entries* $\mathbf{v}$ *are i.i.d. binomial random variables with equal probability of* ± 1.

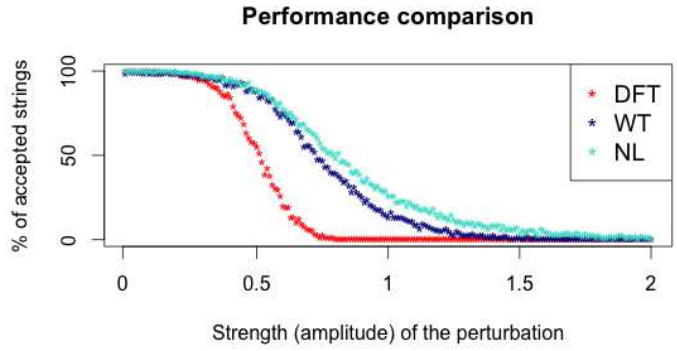


FIGURE 5. *Comparative performance of the non-linearity test*

```
len = 512
count = matrix(0, nrow = 201, ncol = 1)

for(b in 0:200){ for (j in 0:32){ for (h in 1:15){
  sequence = (sign(rnorm(len)+(b/100)*cos((j/10)*c
      (0:(len-1)))) >= 0 )
  NonLin = nl(BooleanFunction(sequence))
  count[b+1] = count[b+1] + (NonLin > 203)
}}}

percent = count * (100/33*15)
plot(percent)
```

.

A quick look at the data (not shown here) suggests that the non-linearity test has some particular "blind spots" by not detecting strong perturbations (large b) for some special values of the frequency parameter j of the equation (4).

3.8. **Concluding remarks.** Above, we have listed different spectral methods of testing for periodic regularities in a sequence of bits. We can see a clear progression from more sensitive and costly (in terms of time and memory) to less sensitive but lighter, and it is up to the engineers to find where to strike the balance.

We note that the DFT method taken directly from [1, sec.3.6] performs significantly less well than our other methods

The combination of the DFT and the maximum absolute value statistics

$$(v_n)_{n=1}^{512} \overset{DFT}{\to} (F_k)_{k=0}^{511} \to \max_k |F_k|$$

remains to be tested.

We also reiterate our opinion that a FFT transform bases on 2-byte real arithmetic should also be tested.

References

[1] A.Rukhin et al,. A statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical Report SP 800–22, National Institude of Standards and Technology, May 15 2011. Special Publication.

[2] Jean Berstel and Juhani Karhumäki. Combinatorics on words-a tutorial. *Current trends in theoretical computer science. The challenge of the new century*, 2:415–475, 2004.

[3] B.Güner, M.Frankford, and J.T.Johnson. A study of the shapiro-wilk test for the detection of pulsed sinusoidal radio frequency interference. *IEEE Transactions on Geoscience and Remote Sensing*, 47(6):1745 – 1751, 2009.

[4] E.O.Brigham. *The Fast Fourier transform and its applications*. Prentice-Hall, 1988.

[5] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

[6] P.Royston. Approximating the shapiro-wilk w-test for non-normality. *Statistics and Computing*, 2:117–119, 1992.

[7] François Rodier. Asymptotic nonlinearity of boolean functions. *Designs, Codes and Cryptography*, 40(1):59–70, 2006.

[8] K.-U. Schmidt. Nonlinearity measures of random Boolean functions. *ArXiv e-prints*, August 2013.

(1) Aix-Marseille Université, Institut de Mathématiques de Marseille, case 907, Marseille 13288 cedex 9, France

E-mail address: florian.cauller@etu.univ-amu.fr

(2) Institut des Hautes Études Scientifiques, Le Bois-Marie, 35 route de Chartres, Bures-sur-Yvette, 91440, France

E-mail address: getmanenko@ihes.fr

(3) Université Paris-Sud, Laboratoire de Mathématiques d'Orsay, Orsay 91405 cedex, France

E-mail address: vito.mandorino@math.u-psud.fr

(4) Université Paris-Est Marne-la-Vallée, Laboratoire d'Informatique Gaspard Monge, 5 Boulevard Descartes, Champs-sur-Marne, Marne-la-Vallée 77454 cedex 2,, France.

E-mail address: vong@univ-mlv.fr